

Re: Byte array to string and back – newbie question

Source:

<http://www.derkeiler.com/Newsgroups/microsoft.public.dotnet.framework.aspnet.security/2004-02/0095.html>

From: Craig (*cscheets_at_remoovdis.kc.rr.com*)

Date: 02/05/04

Date: Thu, 5 Feb 2004 13:47:45 -0600

Actually, this might help you a LOT. I didn't write this, and I don't seem to have bookmarked where it came from, but this class works GREAT and it's SUPER documented so you can dissect what he's doing. It includes functions to encode and decode with any options you can dream. Just snip below the line and you're set...

```
using System;

using System.IO;

using System.Security.Cryptography;

namespace perfClientConfig
{
    /// <summary>
    /// Summary description for EncDec.
    /// </summary>
    //
    // Sample encrypt/decrypt functions
    // Parameter checks and error handling are ommited for better readability
    //
    public class EncDec
    {
```

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
// Encrypt a byte array into a byte array using a key and an IV
public static byte[] Encrypt(byte[] clearData, byte[] Key, byte[] IV)
{
    // Create a MemoryStream that is going to accept the encrypted bytes
    MemoryStream ms = new MemoryStream();

    // Create a symmetric algorithm.

    // We are going to use Rijndael because it is strong and available on all
    platforms.

    // You can use other algorithms, to do so substitute the next line with
    something like

    // TripleDES alg = TripleDES.Create();

    Rijndael alg = Rijndael.Create();

    // Now set the key and the IV.

    // We need the IV (Initialization Vector) because the algorithm is operating
    in its default

    // mode called CBC (Cipher Block Chaining). The IV is XORed with the first
    block (8 byte)

    // of the data before it is encrypted, and then each encrypted block is
    XORed with the

    // following block of plaintext. This is done to make encryption more
    secure.

    // There is also a mode called ECB which does not need an IV, but it is much
    less secure.

    alg.Key = Key;

    alg.IV = IV;

    // Create a CryptoStream through which we are going to be pumping our data.

    // CryptoStreamMode.Write means that we are going to be writing data to the
    stream

    // and the output will be written in the MemoryStream we have provided.
```

Re: Byte array to string and back – newbie question

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
CryptoStream cs = new CryptoStream(ms, alg.CreateEncryptor(),
CryptoStreamMode.Write);

// Write the data and make it do the encryption

cs.Write(clearData, 0, clearData.Length);

// Close the crypto stream (or do FlushFinalBlock).

// This will tell it that we have done our encryption and there is no more
data coming in,

// and it is now a good time to apply the padding and finalize the
encryption process.

cs.Close();

// Now get the encrypted data from the MemoryStream.

// Some people make a mistake of using GetBuffer() here, which is not the
right way.

byte[] encryptedData = ms.ToArray();

return encryptedData;

}

// Encrypt a string into a string using a password

// Uses Encrypt(byte[], byte[], byte[])

public static string Encrypt(string clearText, string Password)

{

// First we need to turn the input string into a byte array.

byte[] clearBytes = System.Text.Encoding.Unicode.GetBytes(clearText);

// Then, we need to turn the password into Key and IV

// We are using salt to make it harder to guess our key using a dictionary
attack –

// trying to guess a password by enumerating all possible words.

PasswordDeriveBytes pdb = new PasswordDeriveBytes(Password,

new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65,
0x64, 0x65, 0x76});
```

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
// Now get the key/IV and do the encryption using the function that accepts  
byte arrays.
```

```
// Using PasswordDeriveBytes object we are first getting 32 bytes for the  
Key
```

```
// (the default Rijndael key length is 256bit = 32bytes) and then 16 bytes  
for the IV.
```

```
// IV should always be the block size, which is by default 16 bytes (128  
bit) for Rijndael.
```

```
// If you are using DES/TripleDES/RC2 the block size is 8 bytes and so  
should be the IV size.
```

```
// You can also read KeySize/BlockSize properties off the algorithm to find  
out the sizes.
```

```
byte[] encryptedData = Encrypt(clearBytes, pdb.GetBytes(32),  
pdb.GetBytes(16));
```

```
// Now we need to turn the resulting byte array into a string.
```

```
// A common mistake would be to use an Encoding class for that. It does not  
work
```

```
// because not all byte values can be represented by characters.
```

```
// We are going to be using Base64 encoding that is designed exactly for  
what we are
```

```
// trying to do.
```

```
return Convert.ToBase64String(encryptedData);
```

```
}
```

```
// Encrypt bytes into bytes using a password
```

```
// Uses Encrypt(byte[], byte[], byte[])
```

```
public static byte[] Encrypt(byte[] clearData, string Password)
```

```
{
```

```
// We need to turn the password into Key and IV.
```

```
// We are using salt to make it harder to guess our key using a dictionary  
attack –
```

```
// trying to guess a password by enumerating all possible words.
```

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
PasswordDeriveBytes pdb = new PasswordDeriveBytes(Password,

new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65,
0x64, 0x65, 0x76});

// Now get the key/IV and do the encryption using the function that accepts
byte arrays.

// Using PasswordDeriveBytes object we are first getting 32 bytes for the
Key

// (the default Rijndael key length is 256bit = 32bytes) and then 16 bytes
for the IV.

// IV should always be the block size, which is by default 16 bytes (128
bit) for Rijndael.

// If you are using DES/TripleDES/RC2 the block size is 8 bytes and so
should be the IV size.

// You can also read KeySize/BlockSize properties off the algorithm to find
out the sizes.

return Encrypt(clearData, pdb.GetBytes(32), pdb.GetBytes(16));

}

// Encrypt a file into another file using a password

public static void Encrypt(string fileIn, string fileOut, string Password)

{

// First we are going to open the file streams

FileStream fsIn = new FileStream(fileIn, FileMode.Open, FileAccess.Read);

FileStream fsOut = new FileStream(fileOut, FileMode.OpenOrCreate,
FileAccess.Write);

// Then we are going to derive a Key and an IV from the Password and create
an algorithm

PasswordDeriveBytes pdb = new PasswordDeriveBytes(Password,

new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65,
0x64, 0x65, 0x76});

Rijndael alg = Rijndael.Create();

alg.Key = pdb.GetBytes(32);
```

Re: Byte array to string and back – newbie question

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
alg.IV = pdb.GetBytes(16);

// Now create a crypto stream through which we are going to be pumping data.

// Our fileOut is going to be receiving the encrypted bytes.

CryptoStream cs = new CryptoStream(fsOut, alg.CreateEncryptor(),
CryptoStreamMode.Write);

// Now will initialize a buffer and will be processing the input file
in chunks.

// This is done to avoid reading the whole file (which can be huge) into
memory.

int bufferLen = 4096;

byte[] buffer = new byte[bufferLen];

int bytesRead;

do

{

// read a chunk of data from the input file

bytesRead = fsIn.Read(buffer, 0, bufferLen);

// encrypt it

cs.Write(buffer, 0, bytesRead);

} while(bytesRead != 0);

// close everything

cs.Close(); // this will also close the unrelying fsOut stream

fsIn.Close();

}

// Decrypt a byte array into a byte array using a key and an IV

public static byte[] Decrypt(byte[] cipherData, byte[] Key, byte[] IV)

{

// Create a MemoryStream that is going to accept the decrypted bytes
```

Re: Byte array to string and back – newbie question

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
MemoryStream ms = new MemoryStream();

// Create a symmetric algorithm.

// We are going to use Rijndael because it is strong and available on all
platforms.

// You can use other algorithms, to do so substitute the next line with
something like

// TripleDES alg = TripleDES.Create();

Rijndael alg = Rijndael.Create();

// Now set the key and the IV.

// We need the IV (Initialization Vector) because the algorithm is operating
in its default

// mode called CBC (Cipher Block Chaining). The IV is XORed with the first
block (8 byte)

// of the data after it is decrypted, and then each decrypted block is XORed
with the previous

// cipher block. This is done to make encryption more secure.

// There is also a mode called ECB which does not need an IV, but it is much
less secure.

alg.Key = Key;

alg.IV = IV;

// Create a CryptoStream through which we are going to be pumping our data.

// CryptoStreamMode.Write means that we are going to be writing data to the
stream

// and the output will be written in the MemoryStream we have provided.

CryptoStream cs = new CryptoStream(ms, alg.CreateDecryptor(),
CryptoStreamMode.Write);

// Write the data and make it do the decryption

cs.Write(cipherData, 0, cipherData.Length);

// Close the crypto stream (or do FlushFinalBlock).
```

Re: Byte array to string and back – newbie question

```
// This will tell it that we have done our decryption and there is no more
data coming in,

// and it is now a good time to remove the padding and finalize the
decryption process.

cs.Close();

// Now get the decrypted data from the MemoryStream.

// Some people make a mistake of using GetBuffer() here, which is not the
right way.

byte[] decryptedData = ms.ToArray();

return decryptedData;

}

// Decrypt a string into a string using a password

// Uses Decrypt(byte[], byte[], byte[])

public static string Decrypt(string cipherText, string Password)

{

// First we need to turn the input string into a byte array.

// We presume that Base64 encoding was used

byte[] cipherBytes = Convert.FromBase64String(cipherText);

// Then, we need to turn the password into Key and IV

// We are using salt to make it harder to guess our key using a dictionary
attack –

// trying to guess a password by enumerating all possible words.

PasswordDeriveBytes pdb = new PasswordDeriveBytes(Password,

new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65,
0x64, 0x65, 0x76});

// Now get the key/IV and do the decryption using the function that accepts
byte arrays.

// Using PasswordDeriveBytes object we are first getting 32 bytes for the
Key
```

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
// (the default Rijndael key length is 256bit = 32bytes) and then 16 bytes
for the IV.

// IV should always be the block size, which is by default 16 bytes (128
bit) for Rijndael.

// If you are using DES/TripleDES/RC2 the block size is 8 bytes and so
should be the IV size.

// You can also read KeySize/BlockSize properties off the algorithm to find
out the sizes.

byte[] decryptedData = Decrypt(cipherBytes, pdb.GetBytes(32),
pdb.GetBytes(16));

// Now we need to turn the resulting byte array into a string.

// A common mistake would be to use an Encoding class for that. It does not
work

// because not all byte values can be represented by characters.

// We are going to be using Base64 encoding that is designed exactly for
what we are

// trying to do.

return System.Text.Encoding.Unicode.GetString(decryptedData);

}

// Decrypt bytes into bytes using a password

// Uses Decrypt(byte[], byte[], byte[])

public static byte[] Decrypt(byte[] cipherData, string Password)

{

// We need to turn the password into Key and IV.

// We are using salt to make it harder to guess our key using a dictionary
attack –

// trying to guess a password by enumerating all possible words.

PasswordDeriveBytes pdb = new PasswordDeriveBytes(Password,

new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65,
0x64, 0x65, 0x76});
```

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
// Now get the key/IV and do the Decryption using the function that accepts
byte arrays.

// Using PasswordDeriveBytes object we are first getting 32 bytes for the
Key

// (the default Rijndael key length is 256bit = 32bytes) and then 16 bytes
for the IV.

// IV should always be the block size, which is by default 16 bytes (128
bit) for Rijndael.

// If you are using DES/TripleDES/RC2 the block size is 8 bytes and so
should be the IV size.

// You can also read KeySize/BlockSize properties off the algorithm to find
out the sizes.

return Decrypt(cipherData, pdb.GetBytes(32), pdb.GetBytes(16));

}

// Decrypt a file into another file using a password

public static void Decrypt(string fileIn, string fileOut, string Password)

{

// First we are going to open the file streams

FileStream fsIn = new FileStream(fileIn, FileMode.Open, FileAccess.Read);

FileStream fsOut = new FileStream(fileOut, FileMode.OpenOrCreate,
FileAccess.Write);

// Then we are going to derive a Key and an IV from the Password and create
an algorithm

PasswordDeriveBytes pdb = new PasswordDeriveBytes(Password,

new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65,
0x64, 0x65, 0x76});

Rijndael alg = Rijndael.Create();

alg.Key = pdb.GetBytes(32);

alg.IV = pdb.GetBytes(16);

// Now create a crypto stream through which we are going to be pumping data.
```

microsoft.public.dotnet.framework.aspnet.security: Re: Byte array to string and back – newbie question

```
// Our fileOut is going to be receiving the Decrypted bytes.

CryptoStream cs = new CryptoStream(fsOut, alg.CreateDecryptor(),
CryptoStreamMode.Write);

// Now will initialize a buffer and will be processing the input file
in chunks.

// This is done to avoid reading the whole file (which can be huge) into
memory.

int bufferLen = 4096;

byte[] buffer = new byte[bufferLen];

int bytesRead;

do

{

// read a chunk of data from the input file

bytesRead = fsIn.Read(buffer, 0, bufferLen);

// Decrypt it

cs.Write(buffer, 0, bytesRead);

} while(bytesRead != 0);

// close everything

cs.Close(); // this will also close the unrelying fsOut stream

fsIn.Close();

}

//

// Testing function

// I am sure you will be able to figure out what it does!

//

public static void testFunc(string[] args)

{
```

Re: Byte array to string and back – newbie question

```
if (args.Length == 0)
{
    string plainText = "This is some plain text";
    string Password = "Password";

    Console.WriteLine("Plain text: \"\" + plainText + "\", Password: \"\" + Password + "\"");

    string cipherText = Encrypt(plainText, Password );

    Console.WriteLine("Encrypted text: " + cipherText);

    string decryptedText = Decrypt(cipherText, Password);

    Console.WriteLine("Decrypted: " + decryptedText);
}
else
{
    Encrypt(args[0], args[0]+".encrypted", "Password");

    Decrypt(args[0]+".encrypted", args[0]+".decrypted", "Password");
}

Console.WriteLine("Done.");
}
}
}
```